Finding Repeats With Fixed Gap

Roman Kolpakov, Gregory Kucherov

Finding Repeats With Fixed Gap *

Roman Kolpakov
French-Russian Institute
for Informatics and Applied Mathematics
Moscow University
119899 Moscow, Russia
e-mail: roman@vertex.inria.msu.ru

Gregory Kucherov
LORIA/INRIA-Lorraine
615, rue du Jardin Botanique
B.P. 101
54602 Villers-lès-Nancy, France
e-mail: kucherov@loria.fr

Abstract

We propose an algorithm for finding in a word all pairs of occurrences of the same subword within a given distance r. The obtained complexity is $O(n \log r + S)$, where S is the size of the output. We also show how the algorithm can be modified in order to find all such pairs of occurrences separated by a given word. The solution uses an algorithm for finding all quasi-squares *in two strings, a problem that generalizes the well-known problem of searching for squares.*

1. Introduction

Repetitions in words are important objects often playing a fundamental role in combinatorial properties of words and their applications to string processing, such as compression [Sto88] or biological sequence analysis [Gus97].

A great deal of work, in word combinatorics and string matching, has been devoted to *contiguous repetitions*, when a fragment is repeated contiguously two or more times [Cro81, Sli83, Cro83, AP83, ML84, ML85, Mai89, Kos94, IMS97, SG98a, KK99b, SG98b, KK99a]. A simplest form of contiguous repetition is a *square* (tandem repeat), which is a subword of the form uu.

On the other hand, some applications bring up the problem of finding subwords repeated in a word in a possibly non-contiguous way. As an example, it is well known that the suffix tree [McC76, Ukk95] allows to easily compute the longest subword occurring at least twice in a word. More about finding repeated subwords in a word can be found in [Gus97].

An intermediate problem, occurring for example in molecular biology applications, consists in finding subwords repeated within some specified distance. This problem has been studied in a recent paper [BLPS99]. More precisely, the problem considered was to find all subwords uvu, where the size of v, called the *gap*, belongs to a specified interval. Using suffix trees together with binary search trees, it has been shown in [BLPS99] that all such *pairs* (of occurrences of u) can be found in time $O(n \log n + S)$, where S is the size of the output.

In this paper, we consider a restricted version of this problem, when v has a *fixed* size, and show that all repeated occurrences of u with a gap equal to r can be found in time $O(n \log r + S)$.

The approach we use is similar to the one used in [Mai89, KK99a] for finding so-called maximal (contiguous) repetitions. It is based on two ideas. The first one is a special factorization of the word. Slightly different definitions of this factorization are known under the name *s-factorization* [Cro83] (*f-factorization* in [CR94]), or Lempel-Ziv factorization [Gus97], because of its use in the well-known Lempel-Ziv compression method [LZ76, ZL77]. In this paper, for presentation purposes, we use the Lempel-Ziv factorization.

The second component of our method is longest common extension functions [ML84]. To illustrate the idea, assume we are given two words u_1, u_2, and want to compute, for each position i of u_2, the length of the longest prefix of u_1 which occurs at position i in u_2. A variation of the Knuth-Morris-Pratt algorithm (see [ML84, CR94]) allows to compute *all* these lengths in time $O(|u_1| + |u_2|)$. This computation, under different variants, appeared to be very useful in several string matching problems [Cro83, ML84, Mai89, KK99a, SG98b].

After giving basic definitions in Section 2, we first consider a problem of finding *quasi-squares* in two words. This problem, which plays an auxiliary role in this paper, generalizes the problem of finding usual squares in a word and is

*Part of this work has been done during the first author's visit of LORIA/INRIA-Lorraine in August-October 1999, within a joint project of the French-Russian A.M.Liapunov Institut of Applied Mathematics and Informatics at Moscow University. A preliminary version of this work has been published as INRIA research report [KK00].

interesting on its own. In Section 3, we propose an efficient solution to this problem. Then, in Section 4, we present an algorithm for finding all repeats with a fixed gap. Finally, in Section 5 we show how this algorithm can be modified to find all repeated subword occurrences with a *fixed word* between them.

2. Definitions

Consider a word $w = a_1...a_n$. $|w|$ denotes the length of w. $w[i..j]$, for $1 \leq i, j \leq n$, denotes the subword $a_i...a_j$ provided that $i \leq j$, and the empty word otherwise. A position i in w is an integer number between 0 and k, associated to the factorization $w = w'w''$, where $|w'| = i$. A subword v of w is said to start (respectively end) at position i if v is a prefix of w'' (respectively suffix of w'). A subword v contains position i if it starts at a position smaller or equal than i, and ends at a position greater or equal than i.

For a set S, $|S|$ denotes the cardinality of S.

Let $r > 0$ be a given integer. An occurrence in w of subword $\alpha = uvu$, where $|u| > 0$ and $|v| = r$, is called an *r-gapped repeat* (for short, *r-repeat*) in w. The first occurrence of u is called the *left root*, and the second the *right root*. For an r-repeat α, the length $|u|$ is denoted $p(\alpha)$.

3. Finding quasi-squares in two words

In this section we consider an auxiliary problem, which however is interesting on its own, as it generalizes the well-known problem of finding all squares in a word.

Assume we are given two words w', w'' of equal length, $|w'| = |w''| = n$, $n \geq 2$. We say that words w', w'' contain a quasi-square iff for some $1 \leq k \leq n$ and $p > 0$ we have $w'[k..k+p-1] = w''[k+p..k+2p-1]$. By analogy to usual squares, p is called the *period* of the quasi-square, and words $w'[k..k+p-1], w''[k+p..k+2p-1]$ are called respectively its *left root* and *right root*.

Example 1 *The pair* $w' = 01100110$ $w'' = 11011100$ *contains five quasi-squares, with roots* 01 *($k = 1$),* 11 *($k = 2$),* 110 *($k = 2$),* 1 *($k = 3$),* 100 *($k = 3$).*

Given two words, the problem is to find all quasi-squares in them. Clearly, this generalizes the problem of finding all squares in a word which corresponds to finding all quasi-squares in two equal words. Recall that finding all squares in a word is a problem which has been extensively studied. Since the number of all squares can be $O(n^2)$, one way is to consider only *primitively-rooted* squares, of which the number is $O(n \log n)$, or other repetitive structures, such as *maximal integer* or *maximal fractional* repetitions[1]. Several algorithms [Cro81, AP83, ML84] allow to find all such structures in time $O(n \log n)$. Each of these algorithms is able to extract all squares in time $O(n \log n + S)$, where S is the number of output squares (see also [SG98a]). On the other hand, Crochemore [Cro83] proposed an algorithm to test, in linear time, if a word contains at least one square. Using the technique proposed in [KK99a], this algorithm can be actually extended to find all squares in time $O(n + S)$. This bound was claimed in [Kos94], and follows from later works [SG98b, KK99a].

Denote $QS(w', w'')$ the set of all quasi-squares of words w', w''. We show that $QS(w', w'')$ can be computed in time $O(n \log n + S)$, where $S = |QS(w', w'')|$. The algorithm we propose is based only on longest common extension functions and does not use suffix tree-like data structures. It is similar to the algorithm of [ML84] for finding all repetitions. An advantage of the proposed solution is that the output quasi-squares are naturally grouped into families of quasi-squares with the same root length and starting at successive positions in the word[2]. In Example 1, the quasi-squares with roots 110 and 100 form such a family, as they both have length 3 and are shifted by one letter one with respect to the other. We will use this feature of the algorithm in Section 5.

Assume $n = 2m$, and denote $QS_m(w', w'')$ the subset of $QS(w', w'')$ consisting of those quasi-squares which contain position m. To prove the bound $O(n \log n + S)$, it is sufficient to show that all quasi-squares from $QS_m(w', w'')$ can be found in time $O(n + |QS_m(w', w'')|)$.

Decompose $QS_m(w', w'')$ into two subsets $QS^l_m(w', w'')$ and $QS^r_m(w', w'')$ containing position m respectively in the left root and the right root. Consider the set $QS^l_m(w', w'')$ ($QS^r_m(w', w'')$ is treated similarly).

Let $w'[k..k+p-1], w''[k+p..k+2p-1]$ be a quasi-square from $QS^l_m(w', w'')$ with period p, that is $k \leq m \leq k+p-1$. Define $LPR(p)$ to be the length of the longest common prefix of words $w'[m+1..n]$ and $w''[m+p+1..n]$, and $LSF(p)$ to be the length of the longest common suffix of $w'[1..m]$ $w''[m+1..m+p]$. From the considered quasi-square, it is easily seen that $LPR(p) + LSF(p) \geq p$ (see Figure 1). Vice versa, if for some $p = 1, ..., m$, $LPR(p) + LSF(p) \geq p$, then there exists a quasi-square with period p from $QS^l_m(w', w'')$. More precisely, the following Lemma holds.

Lemma 1 *For $p = 1, ..., m$, there exists a quasi-square of $QS^l_m(w', w'')$ with period p iff $LPR(p) + LSF(p) \geq p$. When this inequality holds, there is a family of quasi-squares with period p from $QS^l_m(w', w'')$, with the left roots starting at positions $[m - LSF(p)..m + \min\{LPR(p), p\} - p]$.*

To use Lemma 1 as an algorithm for computing $QS^l_m(w', w'')$, we have to compute values

[1] Formal definitions of these notions can be found in [KK99b]

[2] This families are analogous to *runs* of squares in [IMS97, SG98a]

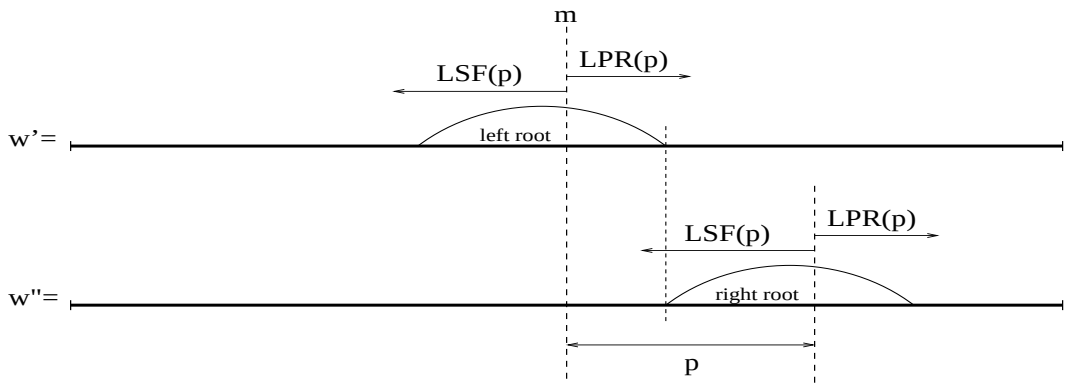

Figure 1. Finding quasi-squares

$LPR(p), LSF(p)$ for $p = 1, ..., m$. All these values can be computed efficiently in time $O(m)$ using a variation of the Knuth-Morris-Pratt string matching algorithm. We refer to [ML84, CR95] for details of how this can be done.

We conclude that the quasi-squares of $QS^l_m(w', w'')$ can be computed in time $O(m + |QS^l_m(w', w'')|)$. Similarly, all quasi-squares of $QS^r_m(w', w'')$ can be computed in time $O(m + |QS^r_m(w', w'')|)$, and therefore all quasi-squares of $QS_m(w', w'')$ in time $O(m + |QS_m(w', w'')|)$. A straightforward divide-and-conquer algorithm gives the running time $O(n \log n + |QS(w', w'')|)$ for finding all quasi-squares in w', w''.

Theorem 1 *The set $QS(w', w'')$ of all quasi-squares in words w', w'' can be found in time $O(n \log n + |QS(w', w'')|)$.*

4. Finding repeats with a fixed gap

We now turn to our main problem – finding all r-repeats in a given word u. We first define the Lempel-Ziv factorization.

Definition 1 *The Lempel-Ziv factorization $w = u_1...u_s$ of a word $w = a_1...a_n$ is recursively defined as follows:*

- $u_1 = a_1$,
- *for $i = 2, ..., s$, $u_i = va$, where v is the longest word, occurring at least twice in $u_1...u_{i-1}v$, and a is the letter following the prefix $u_1...u_{i-1}v$ in w (in other words, u_i is the shortest word which occurs only as a suffix in $u_1...u_{i-1}u_i$).*

Example 2 *The Lempel-Ziv factorization of the word* 1011010110110 *is*

$$1|0|11|010|11011|0$$

The Lempel-Ziv factorization is directly related to the Lempel-Ziv compression algorithm [ZL77] and to the underlying definition of complexity of a string [LZ76]. A salient property of Lempel-Ziv factorization is that it can be computed in time $O(n)$. This can be done using the suffix tree data structure [McC76, Ukk95], developed in the context of string matching applications (see [RPE81]).[3] Conversely, the Lempel-Ziv factorization (and its close relative – the s-factorizaiton [Cro83]) turned out itself to be useful in string matching applications related to the search for repetitions in the word [Mai89, KK99a, SG98b]. This paper gives another example of such an application.

Let $w = a_1...a_n$ be a word of length n. Without loss of generality, we assume that a_n does not occur elsewhere in w. Assume we computed the Lempel-Ziv factorization $w = u_1...u_s$ for w. First, we introduce some notation. Let $e_0, e_1, \ldots, e_{s-1}, e_s$ be the positions delimiting u_i's, that is $e_0 = 0$, and $e_i = |u_1 \ldots u_i|$ for $1 \leq i \leq s$. We also denote $l_i = |u_i|$, $i = 1, ..., s$. For every $i = 1, ..., s-1$, define $\hat{e}_i = e_i + r$, if $l_{i+1} > r$, and $\hat{e}_i = e_{i+1}$, if $l_{i+1} \leq r$. To simplify the presentation, we assume that $w[i] = @$ for $i \leq 0$, where @ is a letter not belonging to the alphabet.

Let us split the set of all r-repeats into the set R' of those r-repeats which contain positions $e_1, ..., e_{s-1}$ and the set R'' of the remaining r-repeats. (Obviously, an r-repeat cannot contain e_s because of the assumption about the last letter, and if it contains e_0, it also contains e_1.) We now concentrate on the r-repeats of R', and further split R' into (disjoint) subsets R'_i, $i = 1, ..., s-1$, so that R'_i consists of those r-repeats which contain e_i but don't contain e_{i+1}. Furthermore, each R'_i is split into the following subsets:

(a) $\alpha \in R^{lrt}_i$ iff the left root of α contains e_i,

(b) $\alpha \in R^{rrt}_i$ iff the right root of α contains e_i,

[3] We note that the Lempel-Ziv factorization can also be computed in linear time with the DAWG data structure [BBH+85, Cro86]

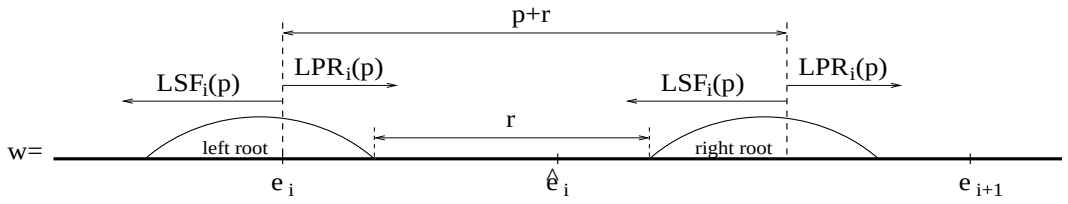

Figure 2. Case (a)

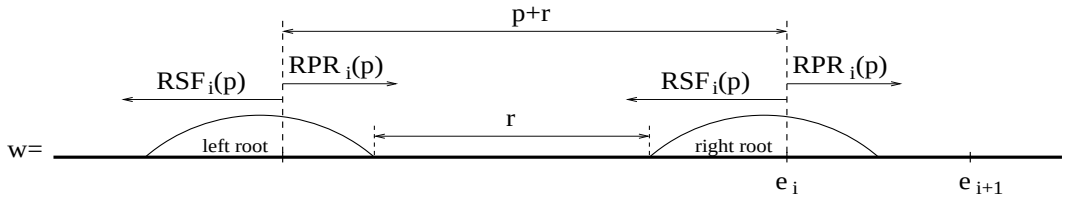

Figure 3. Case (b)

(c) $\alpha \in R_i^{mrt}$ iff the right root of α contains $\hat{e}_i$, but does not contain e_i,

(d) $\alpha \in R_i^{mid}$ iff the right root of α contains neither e_i, nor $\hat{e}_i$.

Cases (a) and (b) cover the situation when e_i is contained in the left (respectively right) root of α. Otherwise, e_i is contained in the gap between the roots. Cases (c) and (d) distinguish whether the right root contains $\hat{e}_i$ or not (note that (c) is not possible if $\hat{e}_i = e_{i+1}$, as α does not contain e_{i+1}).

We now consider separately r-repeats belonging to each of the cases (a)-(d) and show how to find them.

(a) Finding r-repeats of R_i^{lrt}. Let $\alpha \in R_i^{lrt}$ and $p(\alpha) = p$. Since α does not contain e_{i+1}, then $r + p < l_{i+1}$, and therefore $p \leq l_{i+1} - 1 - r$. In particular, R_i^{lrt} is empty whenever $l_{i+1} \leq r + 1$. Assume now that $l_{i+1} > r + 1$.

Define $LPR_i(p)$ to be the length of the longest common prefix of u_{i+1} and $u_{i+1}[r + p + 1..l_{i+1} - 1]$, and $LSF_i(p)$ to be the length of the longest common suffix of $u_{i+1}[r + 1..r + p]$ and the suffix of $u_1 \ldots u_i$ of length p. From the r-repeat α, it is easily seen that $LSF_i(p) + LPR_i(p) \geq p$. The situation is depicted on Figure 2.

Conversely, if for some $p = 1..l_{i+1} - 1 - r$, $LSF_i(p) + LPR_i(p) \geq p$, then there exists a family of r-repeats of R_i^{lrt} with the root length p, starting at positions $[e_i - LSF_i(p)..e_i + \min\{0, LPR_i(p) - p\}]$.

We summarize the above in the following lemma.

Lemma 2 *There exists an r-repeat $w \in R_i^{lrt}$ with root length p ($p \in [1..l_{i+1} - 1 - r]$) iff $LSF_i(p) + LPR_i(p) \geq p$. When this inequality holds, all such r-repeats start at positions $[e_i - LSF_i(p)..e_i + \min\{0, LPR_i(p) - p\}]$.*

Lemma 2 suggests a method of computing R_i^{lrt}. Compute the longest common extension functions $LSF_i(p)$ and $LPR_i(p)$ for all $p = 1..l_{i+1} - 1 - r$. This computation can be done in time linear on the length of involved words, that is in time $O(l_{i+1})$, using the Knuth-Morris-Pratt technique (see Section 3). Then all r-repeats of R_i^{lrt} can be output using Lemma 2. The whole computation takes time $O(l_{i+1} + |R_i^{lrt}|)$.

(b) Finding r-repeats of R_i^{rrt}. Consider $\alpha \in R_i^{rrt}$ with $p(\alpha) = p$. From Definition 1 of Lempel-Ziv factorization it follows that the right root of α starts to the right of e_{i-1}. On the other hand, from the definition of R_i^{rrt}, it ends to the left of e_{i+1}. Therefore, $0 < p \leq l_i + l_{i+1} - 2$.

We proceed similarly to case (a), and define longest common extension functions $RPR_i(p)$ and $RSF_i(p)$ for $p = 1..l_i + l_{i+1} - 2$. $RPR_i(p)$ is defined as the length of the longest common prefix of $u_{i+1}[1..l_{i+1} - 1]$ and $w[e_i - r - p + 1..e_i - r]$, and $RSF_i(p)$ as the length of the longest common suffix of $u_i[2..l_i]$ and the suffix of $w[1..e_i - r - p]$ of length $l_i - 1$. Similarly to case (a), the following Lemma holds (see Figure 3).

Lemma 3 *There exists an r-repeat $\alpha \in R_i^{rrt}$ with root length p ($p \in [1..l_i + l_{i+1} - 2]$) iff $RPR_i(p) + RSF_i(p) \geq p$. When this inequality holds, the right roots of all such r-repeats start at positions $[e_i - \min\{RSF_i(p), p\}..e_i + RPR_i(p) - p]$.*

Again, functions RPR_i and RSF_i can be computed in time linear in the length of involved words, that is in time $O(l_i + l_{i+1})$. Therefore, all r-repeats of R_i^{rrt} can be reported in time $O(l_i + l_{i+1} + |R_i^{rrt}|)$.

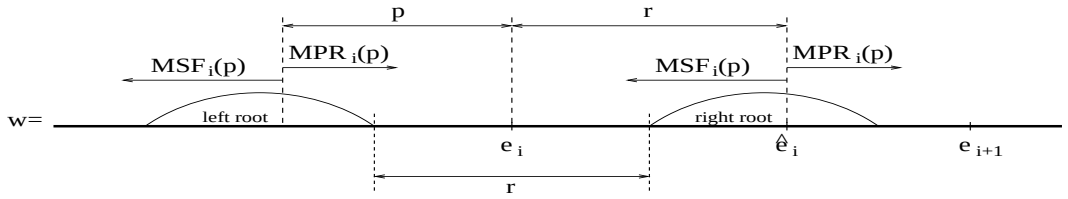

Figure 4. Case (c)

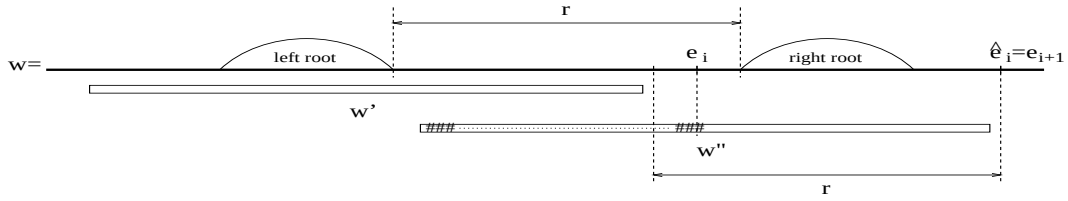

Figure 5. Case (d)

(c) Finding r-repeats of R_i^{mrt}. Note that this case is defined only when $\hat{e}_i < e_{i+1}$, that is when $l_{i+1} > r$. Consider $\alpha \in R_i^{mrt}$ with $p(\alpha) = p$. The right root of α lies inside $u_{i+1}[2..l_{i+1} - 1]$, and therefore $p < l_{i+1} - 1$.

Using the same approach, we define $MPR_i(p)$ to be the length of the longest prefix of $u_{i+1}[r + 1..l_{i+1} - 1]$ and $w[e_i - p + 1..e_i]$, and $MSF_i(p)$ to be the length of the longest suffix of $u_{i+1}[2..r]$ and the suffix of $w[1..e_i - p]$ of length $r - 1$. The following Lemma holds (see Figure 4).

Lemma 4 *There exists an r-repeat $\alpha \in R_i^{mrt}$ with root length p ($p \in [1..l_{i+1} - 2]$) iff $MPR_i(p) + MSF_i(p) \geq p$. When this inequality holds, the right roots of all such r-repeats start at positions $[e_i + r - \min\{MSF_i(p), p\}..e_i + r + MPR_i(p) - p]$.*

Functions MPR_i and $MSF_i(p)$ can be computed in time $O(l_{i+1})$ and all r-repeats of R_i^{mrt} can be reported in time $O(l_{i+1} + |R_i^{mrt}|)$.

(d) Finding r-repeats of R_i^{mid}. Consider now $\alpha \in R_i^{mid}$ with $p = p(\alpha)$. Denote $m_i = \hat{e}_i - e_i = \min\{r, l_{i+1}\}$. The right root of α lies inside $u_{i+1}[2..m_i - 1]$, and therefore $p \leq m_i - 2$.

This case differs from cases (a)-(c) in that we cannot *a priori* select a position contained in the right (or left) root of α. Therefore, we cannot apply directly the technique of longest common extension functions. We reduce this case to the problem of finding quasi-squares, considered in Section 3.

Since the start position of the right root is contained in the word $w[e_i + 2..e_i + m_i - 1]$, the end position of the left root is contained in the word $w[e_i + 2 - r..e_i + m_i - 1 - r]$. Since $p \leq m_i - 2$, the left root of α is contained in the word $w' = w[e_i - r - m_i + 4..e_i + m_i - 1 - r]$. The length of w' is $2m_i - 4$. Let # be another fresh letter. Denote by w'' the word $\underbrace{\#...\#}_{m_i - 2} u_i[2..m_i - 1]$. The construction is illustrated in Figure 5 (case $l_{i+1} < r$).

Lemma 5 *There exists an r-repeat $\alpha \in R_i^{mid}$ iff there exists a quasi-square in words w', w''. Each such quasi-square corresponds to an r-repeat $\alpha \in R_i^{mid}$.*

Therefore, there is a one-to-one correspondence between the set R_i^{mid} and the set of quasi-squares in the words w', w'' constructed above. Moreover, a quasi-square with the left root starting at position j in w' corresponds to an r-repeat starting at position $(e_i - r - m_i + 3 + j)$ in w.

By Theorem 1, all those quasi-squares can be found in time $O(m_i \log m_i)$. We conclude that all r-repeats of R_i^{mid} can be reported in time $O(m_i \log m_i + |R_i^{mid}|)$, which, using $m_i = \min\{r, l_{i+1}\}$, we estimate as $O(l_{i+1} \log r + |R_i^{mid}|)$.

Putting together cases (a)-(d), all r-repeats of R_i' can be found in time $O(l_i) + O(l_{i+1} \log r) + O(|R_i'|)$. Summing up over all $i = 1..s$, we obtain that all r-repeats of R' can be found in time $O(n \log r + |R'|)$.

Finding r-repeats of R'' can be done using a technique similar to the one used in [KK99a]. The key observation here is that each r-repeat of R'' occurs inside some factor u_i (i.e. does not contain positions e_i and e_{i+1}). By definition of the factorization, each such r-repeat is a copy of another r-repeat occurring to the left. When constructing the Lempel-Ziv factorization, we can store, for each factor $u_i = va$, a reference to an occurrence of v to the left (see Definition 1). After finding all r-repeats of R', we sort

them, using basket sort, in increasing order of their start position and, for each start position, in increasing order of their root length. Then we process all factors from left to right and for each factor $u_i = va$, copy corresponding earlier found r-repeats occurring in the referenced copy of v. We refer the reader to [KK99a] for full details. The running time of this stage is $O(n + |R''|)$.

We conclude with the final result.

Theorem 2 *The set R of all r-repeats in a word w can be found in time $O(n \log r + |R|)$.*

We end this section by noting that when $r = 0$ (that is, usual squares are looked for), only cases (a),(b) remain to be dealt with. The algorithm we obtain is actually the algorithm of Crochemore [Cro83] allowing to find, in linear time, all squares containing factor borders, augmented with the technique of [KK99a] allowing to find the remaining squares (cf Section 3). Thus, we obtain an $O(n + S)$ algorithm for finding all squares. The same algorithm works for $r = 1$, since in this case too, cases (a),(b) cover all possible relative positions of 1-repeats and factor borders.

5. Finding r-repeats with a fixed gap word

The algorithm presented in Section 4 can be modified in order to find all r-repeats with a fixed word between the two roots. Assume v is a fixed word of length r. Denote by R_v the set of r-repeats of the form uvu, where $|u| \geq 1$. We show that all those repeats can be found in time $O(n \log r + |R_v|)$. To do that, we first find, using any linear-time string matching algorithm (for example, the Knuth-Morris-Pratt algorithm) all start occurrences of v in w. For each position i of v, we compute the position $NEXT(i)$, defined as the nearest start position of v strictly to the right of i.

From the algorithm of Section 4 for finding the set R', it should be clear that all the r-repeats of R' can be represented by $O(n \log r)$ families each consisting of r-repeats with a given root length and starting at all positions from a given interval. In other words, each family can be specified by an interval $[i..j]$ and a number p, and encodes all r-repeats with root length p starting at positions from $[i..j]$.

From this specification, using function $NEXT(i)$, we can easily extract all r-repeats of R_v in time proportional to the number of those. For that, we first assume that each family is specified by an interval of *end positions* of the left root (as the root length p is known for each family, the translation can be trivially computed by just adding p to the interval of start positions). Then we process all the families and extract from each interval those positions which are start positions of an occurrence of v. Using function $NEXT$, this can be easily done in time proportional to the number of such positions.

After processing all families, we have found all r-repeats from the set $R'_v = R_v \cap R'$ in time $O(n \log r + |R'_v|)$. Then, using a procedure for finding r-repeats from R'', described in Section 4, we find all r-repeats from $R''_v = R_v \cap R''$ in time $O(n + |R''_v|)$. As $R_v = R'_v \cup R''_v$, all r-repeats from R_v are found in time $O(n \log r + |R_v|)$.

6. Conclusions

An interesting natural question is whether all r-repeats can be found in time $O(n + |R|)$. The "bottleneck" implying the $\log r$ factor comes from the problem of finding quasi-squares. Can all quasi-squares be found in time $O(n + |QS(w', w'')|)$?

References

[AP83] A. Apostolico and F.P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22(3):297–315, 1983.

[BBH+85] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[BLPS99] G. Brodal, R. Lyngsø, Ch. Pedersen, and J. Stoye. Finding maximal pairs with bounded gap. In M. Crochemore and M. Paterson, editors, *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, volume 1645 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[CR94] M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.

[CR95] M. Crochemore and W. Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13:405–425, 1995.

[Cro81] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12:244–250, 1981.

[Cro83] M. Crochemore. Recherche linéaire d'un carré dans un mot. *Comptes Rendus Acad. Sci. Paris Sér. I Math.*, 296:781–784, 1983.

[Cro86] M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986.

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[IMS97] C.S. Iliopoulos, D. Moore, and W.F. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172:281–291, 1997.

[KK99a] R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 1999 Symposium on Foundations of Computer Science, New York (USA)*. IEEE Computer Society, October 17-19 1999.

[KK99b] R. Kolpakov and G. Kucherov. On maximal repetitions in words. In *Proceedings of the 12-th International Symposium on Fundamentals of Computation Theory, 1999, Iasi (Romania)*, Lecture Notes in Computer Science, August 30 - September 3 1999.

[KK00] Roman Kolpakov and Gregory Kucherov. Finding repeats with fixed gap. Technical Report RR-3901, INRIA, March 2000.

[Kos94] S. R. Kosaraju. Computation of squares in string. In M. Crochemore and D. Gusfield, editors, *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, number 807 in Lecture Notes in Computer Science, pages 146–150. Springer Verlag, 1994.

[LZ76] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory IT-22*, pages 75–81, Jan 1976.

[Mai89] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25:145–153, 1989.

[McC76] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[ML84] M.G. Main and R.J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

[ML85] M.G. Main and R.J. Lorentz. Linear time recognition of square free strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 272–278. Springer Verlag, 1985.

[RPE81] M. Rodeh, V.R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *Journal of the ACM*, 28(1):16–24, Jan 1981.

[SG98a] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. In M. Farach-Colton, editor, *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, number 1448 in Lecture Notes in Computer Science, pages 140–152. Springer Verlag, 1998.

[SG98b] J. Stoye and D. Gusfield. Linear time algorithms for finding and representing all the tandem repeats in a string. Technical Report CSE-98-4, Computer Science Department, University of California, Davis, 1998.

[Sli83] A.O. Slisenko. Detection of periodicities and string matching in real time. *Journal of Soviet Mathematics*, 22:1316–1386, 1983.

[Sto88] J.A. Storer. *Data Compression: Methods and Theory*. Computer Science Press, Rockville, MD, 1988.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory IT-23*, 3:337–343, May 1977.